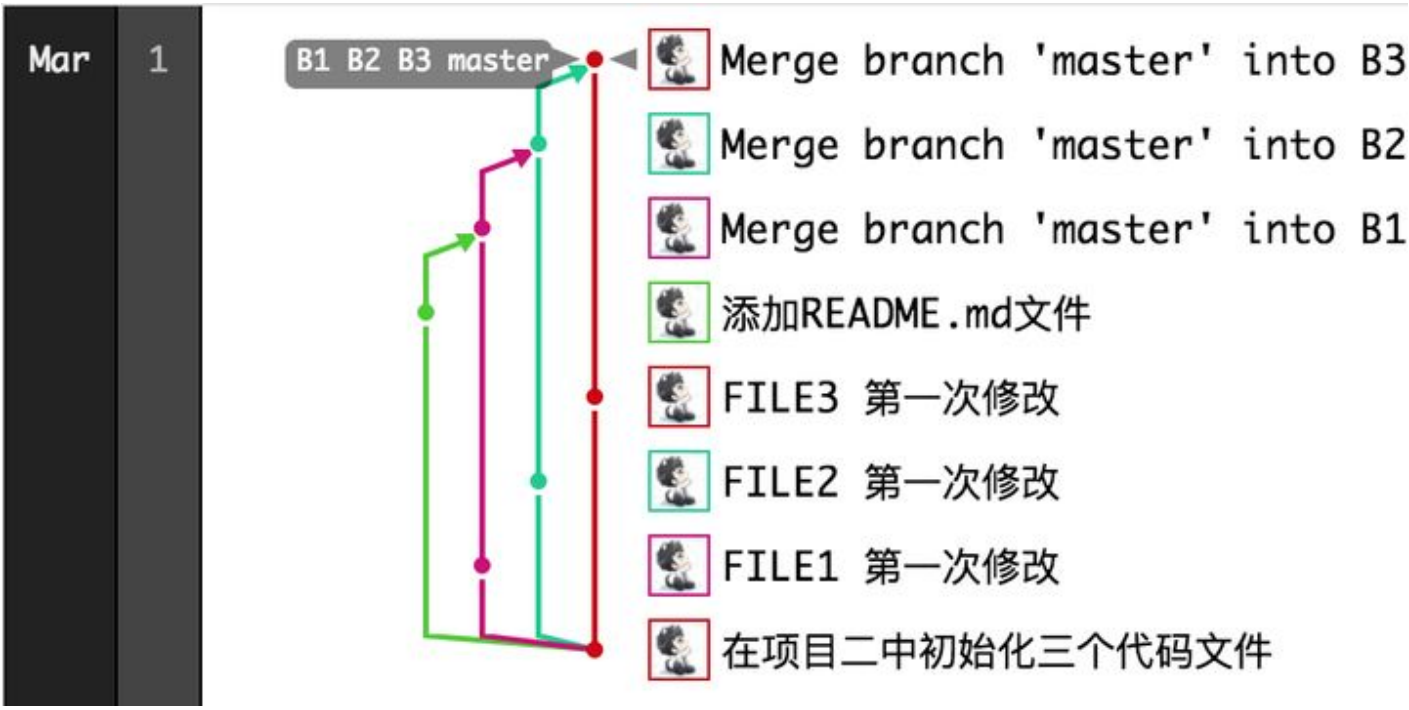
背景

使用GIT这么久了从来没有深层次的研究过，一般情况下，只要会用 `pull`，`commit`，`push` 等几个基本提交命令就可以了，公司的项目分支管理这部分操作一直都是我负责，对于分支的合并我一直都使用 `merge` 操作，也知道还有一个 `rebase` ，但是一直不会用，百度了很多，说的基本都差不多，按照步骤在公司项目里操作，简直就是噩梦，只要 `rebase` 就出现噩梦般的冲突，所以一直不敢用，今天自己捣腾了一番终于领略到一些，不多说直接进入干货。

先来两张合理使用 `rebase` ， `merge` 和只使用 `merge` 的对比图

使用 rebase

使用 merge



使用 rebase 和 merge 的基本原则：

下游分支更新上游分支内容的时候使用 rebase
上游分支合并下游分支内容的时候使用 merge
更新当前分支的内容时一定要使用 --rebase 参数

例如现有上游分支 master，基于 master 分支拉出来一个开发分支 dev，在 dev 上开发了一段时间后要把 master 分支提交的新内容更新到 dev 分支，此时切换到 dev 分支，使用 git rebase master

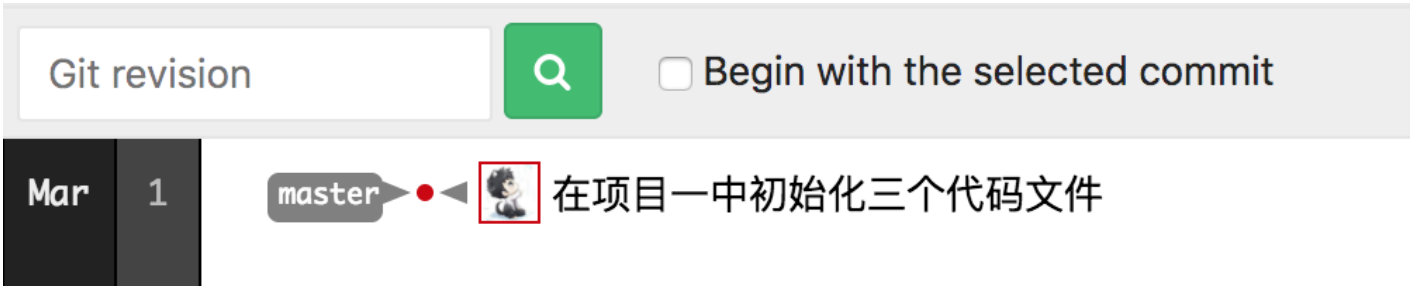
等 dev 分支开发完成了之后，要合并到上游分支 master 上的时候，切换到 master 分支，使用 git merge dev

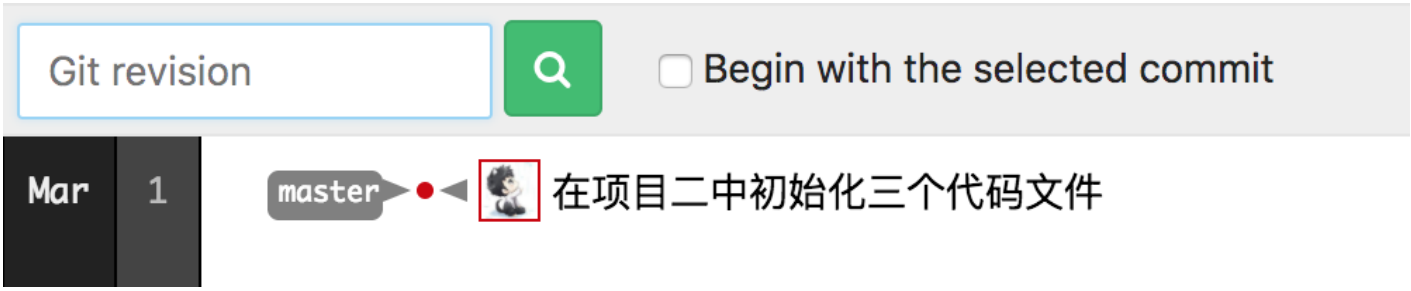
一、创建两个GIT项目， project1 和 project2 ,同时分别加入三个文件并提交 master 分支

```
$ git clone git@gitlab.xpaas.lenovo.com:baiyl3/project1.git
$ cd project1
$ touch file1 file2 file3
$ git add .
$ git commit -m '在项目一中初始化三个代码文件'
$ git push -u origin master

$ git clone git@gitlab.xpaas.lenovo.com:baiyl3/project2.git
$ cd project2
$ touch file1 file2 file3
$ git add .
$ git commit -m '在项目二中初始化三个代码文件'
$ git push -u origin master
```

从代码提交时间轴图看两个项目现在的状态都一致

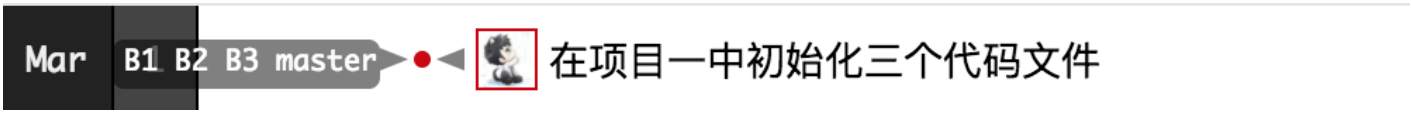


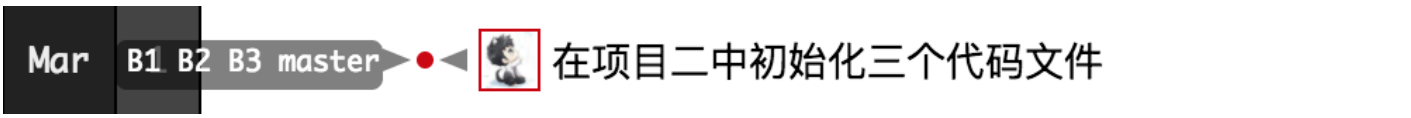


二、分别给两个项目创建三个分支 B1，B2，B3 (* 对应数字)

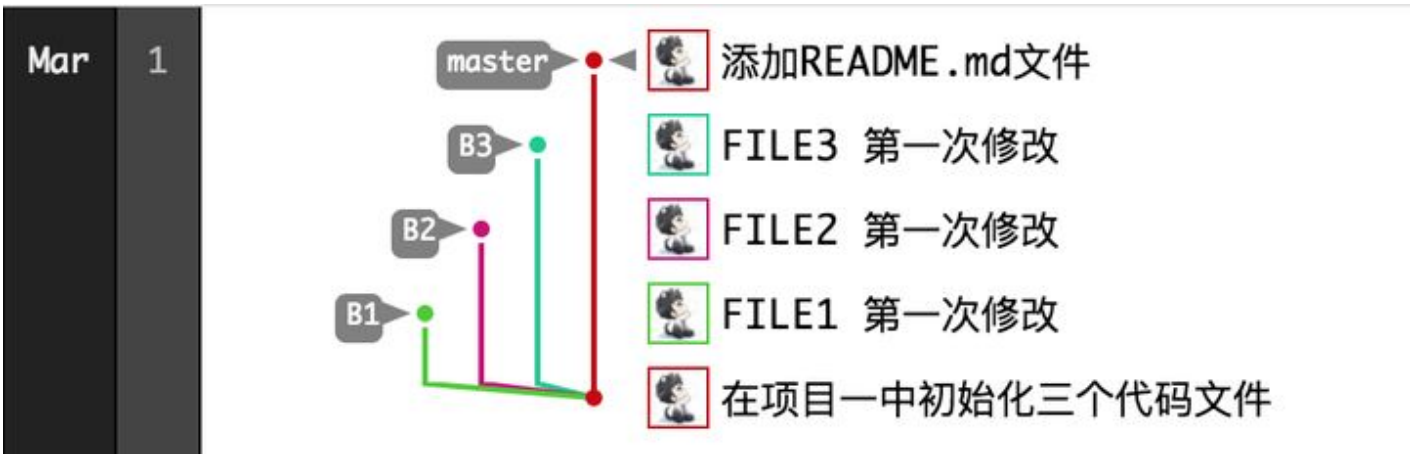
```
$ git checkout -b B*
$ git push origin B*
$ git branch -a
  B1
  B2
* B3
  master
  remotes/origin/B1
  remotes/origin/B2
  remotes/origin/B3
  remotes/origin/master
$ git logs --graph
* 891d1ed<baiyl3> - (HEAD -> B3, origin/master, origin/B3, origin/B2, origin/B1, master
  file1
  file2
  file3
```

从 `git log` 中我们看到 B1，B2，B3 都基于master最新提交点拉出来的三个新分支，如下图从时间轴也可以看出





三、现在我们分别切换分支到 B1，B2，B3，并修改对应的文件 file1，file2，file3，最后切换到 mastet 分支添加一个 README.md 文件，然后再看时间轴：



从上图的结果可以看出，B1, B2, B3, master 四个分支分别在不同的时间点做了代码提交，那么最后一次 master 上做的修改在 B1, B2, B3 三个分支上肯定没有

四、此时在这三个分支上开发的同学发现他们要做的功能要在 master 最新的基础上开发，或者他们也想要 master 上最新的内容，重点来了，现在我们怎么办？方法有两种，一种是使用 `rebase` ，另一种是使用 `merge` ，我们分别在 project1 和 project2 两个项目上使用这两种方式解决这个问题

在项目 project1 使用 rebase

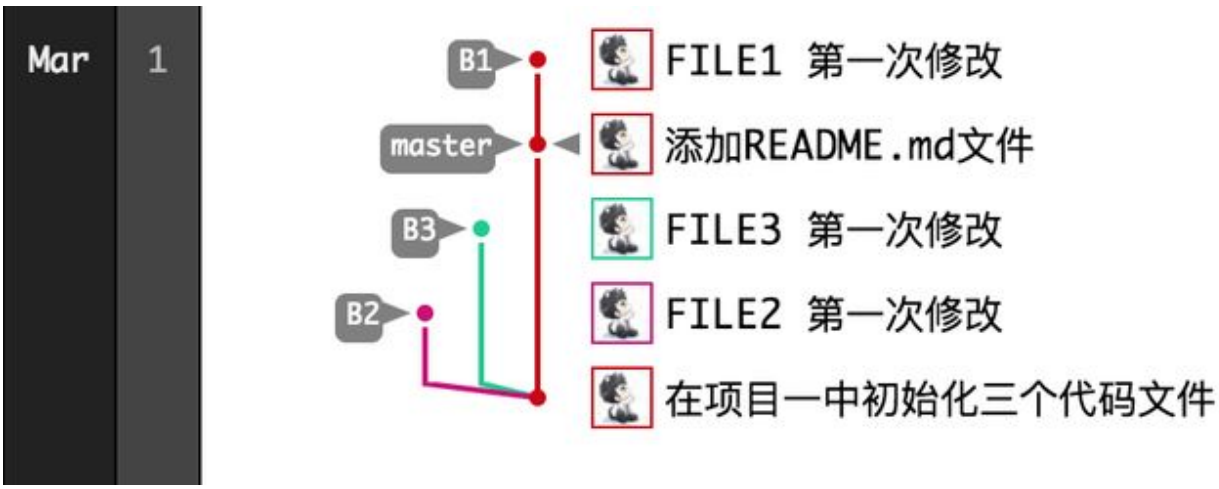
```
$ cd project1
$ git checkout B1
$ git pull origin B1 --rebase
From gitlab.xpaas.lenovo.com:baiyl3/project1
 * branch            B1         -> FETCH_HEAD
Already up-to-date.
Current branch B1 is up to date.

$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: FILE1 第一次修改

$ git push origin B1
To gitlab.xpaas.lenovo.com:baiyl3/project1.git
 ! [rejected]        B1 -> B1 (non-fast-forward)
error: failed to push some refs to 'git@gitlab.xpaas.lenovo.com:baiyl3/project1.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

$ git push origin B1 --force
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 328 bytes | 328.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: To create a merge request for B1, visit:
remote:   http://gitlab.xpaas.lenovo.com/baiyl3/project1/merge_requests/new?merge_reque
remote:
To gitlab.xpaas.lenovo.com:baiyl3/project1.git
 + b3052a0...00032a7 B1 -> B1 (forced update)

$ ls
README.md   file1       file2       file3
```



在上面的过程中，更新代码我使用的是 `git pull origin B1 --rebase` 而不是 `git pull origin B1` 这也是平时使用 rebase 注意的一点， `git pull` 这条命令默认使用了 `--merge` 的方式更新代码，如果你不指定用 `--rebase` ，有的时候就会发现日志里有这样的一次提交 `Merge branch 'dev' of gitlab.xpaas.lenovo.com:liuyy23/lenovo-mbg into dev` 什么？自己分支合并到了自己分支，显然这个没有什么必要，而且在时间轴上也不好看，平白无故多了一条线出来，对于强迫症的我来说看着就难受。。。

还有就是使用 rebase 之后，如果直接使用 `git push origin B1` 发现是不好使的，提示也说明了提交失败的原因，我个人是这么理解的，使用 rebase 之后，master分支上比B1分支上多的修改，直接“插入”到了B1分支修改的内容之后，也就是 master 分支的修改在 B1 分支上重演了一遍，相对远程 B1 分支而言，本地仓库的B1分支的“基底”已经变化了，直接 `push` 是不行的，所以确保没有问题的情况下必须使用 `--force` 参数才能提交，这也就是官方对 `rebase` 作为 “变基” 的解释（个人观点）。

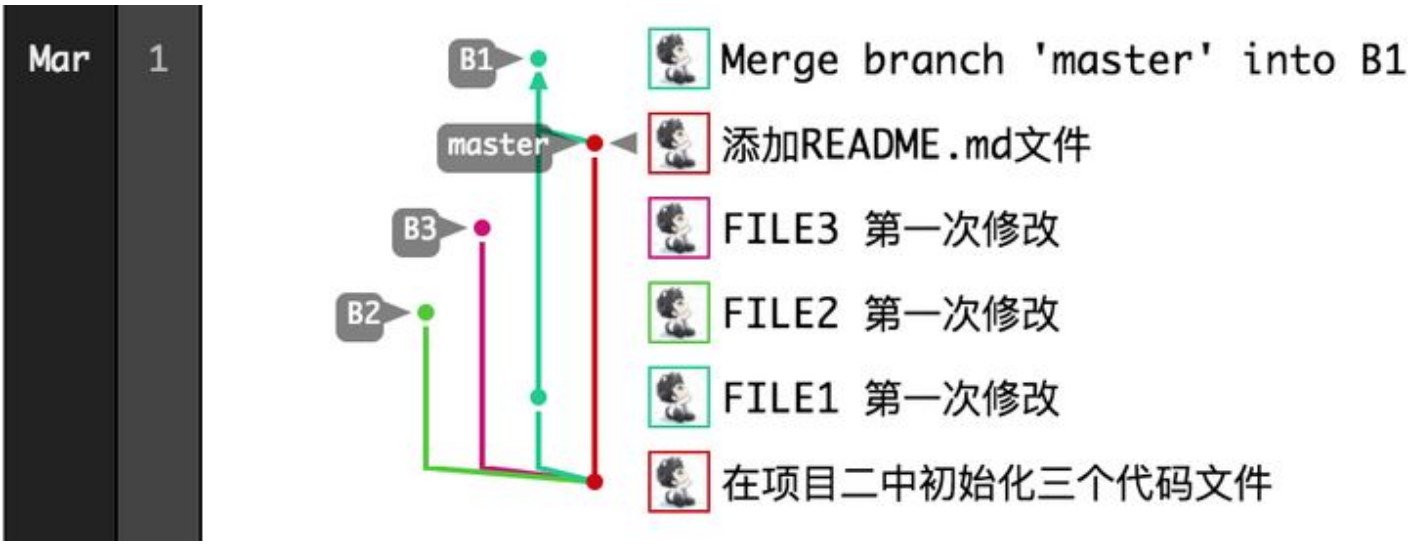
接下来我们接着在 project2 项目上使用 merge 操作

```
$ cd project
$ git pull origin B1
From gitlab.xpaas.lenovo.com:baiyl3/project2
 * branch            B1         -> FETCH_HEAD
Already up-to-date.

$ git merge master
Merge made by the 'recursive' strategy.
 README.md | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md

$ git push origin B1
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 278 bytes | 278.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote:
remote: To create a merge request for B1, visit:
remote:   http://gitlab.xpaas.lenovo.com/baiyl3/project2/merge_requests/new?merge_reque
remote:
To gitlab.xpaas.lenovo.com:baiyl3/project2.git
   d3ea69c..e040c7b  B1 -> B1
```

```
ls
README.md    file1        file2        file3
```

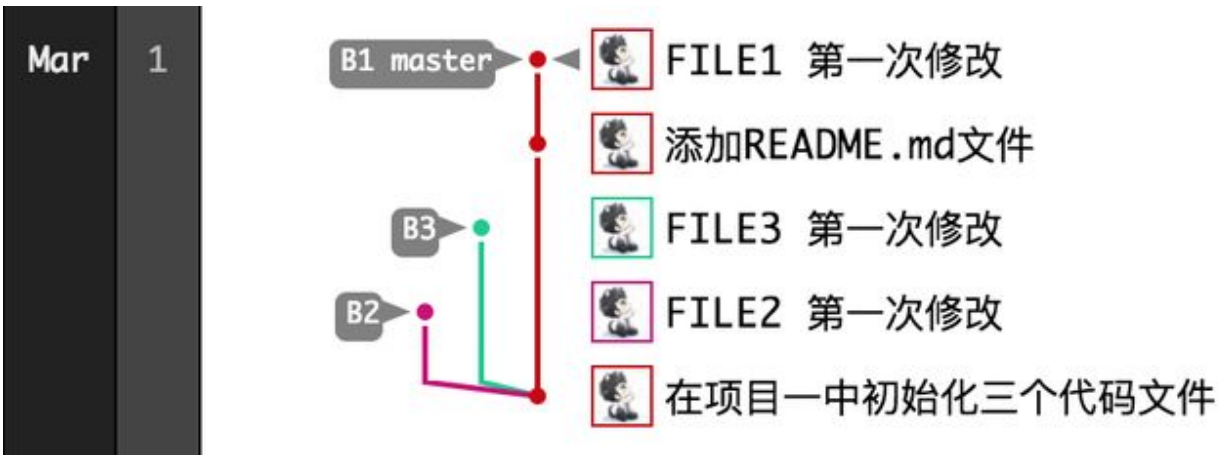


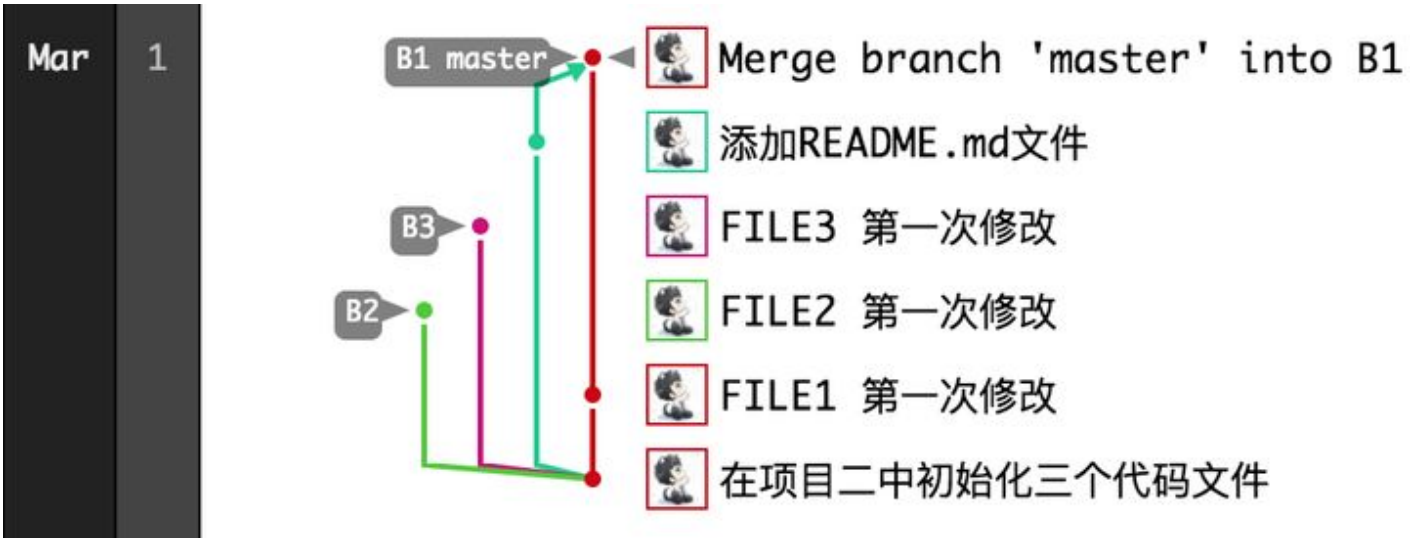
可以看到 merge 之后，在 B1 分支上多出一条合并的log

此时，我们的 B1 分支开发完成了，要合并到 master 分支，根据基本原则，在 master 分支上都使用 `git merge B1` 就可以合并，看下图结果：

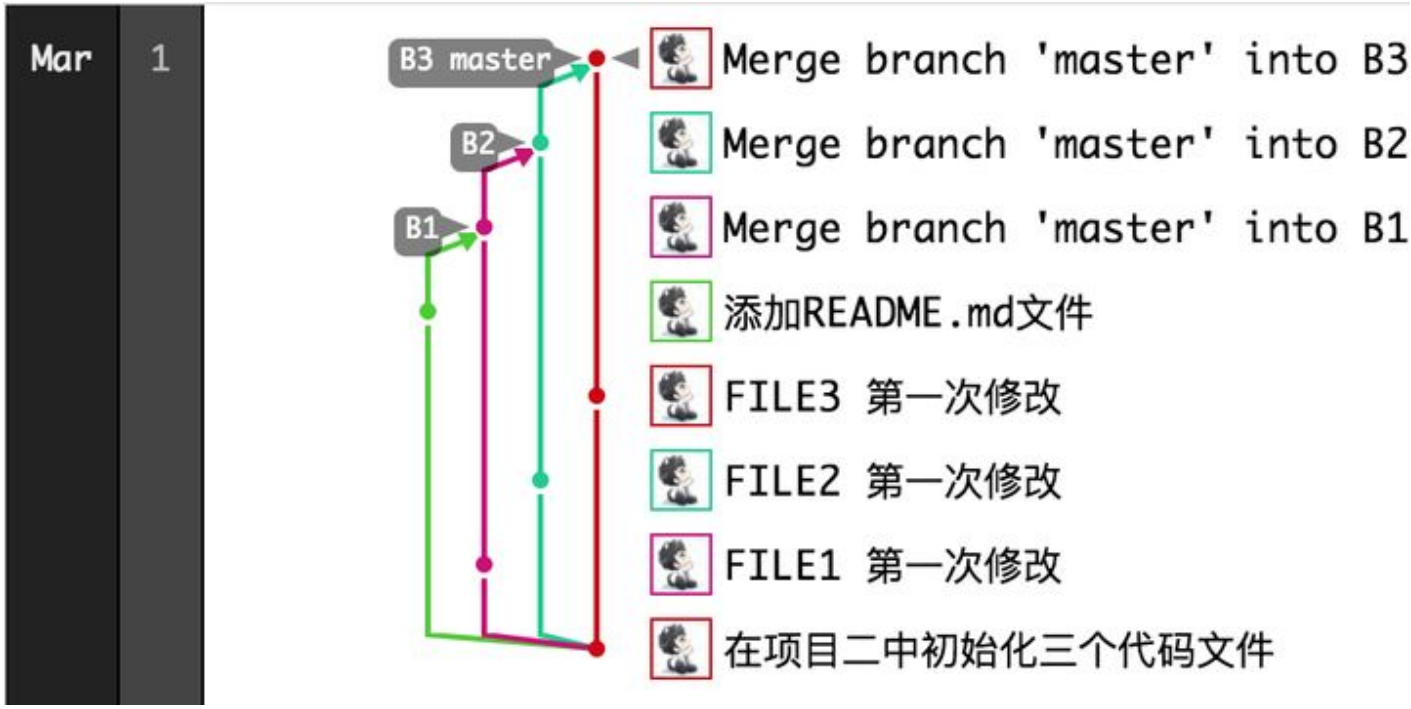
```
$ git checkotu master
$ git merge B1
Updating c782e83..00032a7
Fast-forward
 file1 | 1 +
 1 file changed, 1 insertion(+)

$ git push origin master
```





接下来对 B2，B3 分别在 project1 和 project2 上做相同的操作，我们看结果：



再看看命令行下log的情况

```
$ git logs --graph
* 5826260<baiyl3> - (HEAD -> master, origin/master, origin/B3, B3) FILE3 第一次修改 (6 mi
| file3

* cffcc9a<baiyl3> - (origin/B2, B2) FILE2 第一次修改 (8 minutes ago)|
| file2

* 00032a7<baiyl3> - (origin/B1, B1) FILE1 第一次修改 (87 minutes ago)|
| file1

* c782e83<baiyl3> - 添加README.md文件 (2 hours ago)|
| README.md

* b783e0a<baiyl3> - 在项目一中初始化三个代码文件 (3 hours ago)
  file1
  file2
  file3
git logs --graph
*   bc3f385<baiyl3> - (HEAD -> master, origin/master, origin/B3, B3) Merge branch 'maste
|\
| *   64b4f3d<baiyl3> - (origin/B2, B2) Merge branch 'master' into B2 (5 minutes ago)
| |\
| | *   e040c7b<baiyl3> - (origin/B1, B1) Merge branch 'master' into B1 (35 minutes ago)
| | |\
| | | * 2cedfcb<baiyl3> - 添加README.md文件 (2 hours ago)| | | |
| | | | README.md

| | * | d3ea69c<baiyl3> - FILE1 第一次修改 (2 hours ago)
| | |/  | | |
| | |   file1

| * | 5975eae<baiyl3> - FILE2 第一次修改 (2 hours ago)
| |/  | |
| |   file2

* | 37ec6de<baiyl3> - FILE3 第一次修改 (2 hours ago)
|/  |
|   file3

* 891d1ed<baiyl3> - 在项目二中初始化三个代码文件 (3 hours ago)
  file1
  file2
  file3
```

使用 rebase 就感觉所有人都在同一条直线上开发一样，很干净的log，看着很舒服，而一直使用 merge 的log看起来就很乱，我这只是4个分支的例子，我们项目每周基本都是十几个分支，真的是看起来乱入一团哇。。。

这个例子中的操作都没有出现不同分支修改同一个文件导致冲突的情况，实际开发中这种情况非常多，rebase 的时候出现冲突后 git 也会列出来哪些文件冲突了，等你解决冲突之后使用 `git rebase --continue` 就会继续处理，所以为了避免这种冲突太多，而且不好解决，我们项目中基本都是一个需求就一个分支，而且开发过程中要随时更新上游分支的内容下来，确保在最新的代码基础上开发，这也是 GIT 推荐的方式，尽可能多建分支开发，而不是在一个开发分支上很多人操作，如果是这种情况建议不要用 rebase，另外负责分支合并的人在合并下游分支代码的时候要确保你这个上游分支不要在这段时间内提交代码，有一个方法就是把上游分支 设置 protect

五、注意事项

更新当前分支代码的时候一定要使用 `git pull origin xxx --rebase`
合并代码的时候按照最新分支优先合并为原则
要经常从上游分支更新代码，如果长时间不更新上游分支代码容易出现大量冲突